Volker Roth

Rechnersicherheit, SoSe 21

Übung 03

TutorIn: Oliver Wiese
Tutorium 02
Materialien: Latex, VSC, Skript

19. Mai 2021

---

1 Password models

(a) *Choose your tool to analyze the password set. We recommend python (with pandas)or R but you can choose an other tool. Be aware that most CSV-Tools can not handle a huge file.*

- Wir haben das pandas CSV-Tool benutzt. Da die zu analysierende Datei jedoch keinen seperator zwischen Passworthäufigkeit und Passwort hat, haben wir folgendes versucht:

  1. At first we used a white space as separator, however, this caused some passwords to be left out or incorrectly recorded, which threw several errors as consequence.

     ```
     df = pd.read_csv(p_datafile, sep=' ', header=None,  names=['frequentcys', '
         passwords'])
     ```

  2. After that we used a regex as separator, however, the number of errors only decreased slightly. It also took a lot longer to analyze the file.

     ```
     sep = r'^(\s*[0-9]+)'
     df = pd.read_csv(p_datafile, sep, header=None,  names=['frequentcys', '
         passwords'], engine='python')
     ```

     Snipped of passwords containing errors / not loading with pandas (The passwords in this case are whitespaces):

     ```
     4750:    340
     34329:     61
     50383:     44
     99361:     24
     154363:     16
     231236:     11
     562997:      5
     2459758:      2
     2465704:      1
     ```

3. Then we came up with the idea of using preprocessing before reading the file with pandas. There should now be a comma between each password and the password frequency. The runtime has been improved, but there are several 'password errors' again, because many passwords contain commas.

```
4750:340,
135833:18,,./,./
165384:15,,.,.,.
210363:12,,,,,,,
288009:9,,mnbvcxz
380660:7,,,,,...
453657:6,,fojkiyd
453658:6,,f9tovp
562846:5,,u8;k,l6-
562847:5,,bo9ik
562848:5,,b56okpo
745987:4,,y]]bdk
745988:4,,tgs,ujp;
745989:4,,r87hwtwycv
745990:4,,kpfNojkiyd
745991:4,,kp,bhomN
745992:4,,Iuiy9oN
745993:4,,./;\\'#
745994:4,,.,.,.,.,.
844993:3,nan
```

4. In preprocessing we replaced the comma with a tab. Now there are almost no password errors anymore.

```
#preprocess txt file and open it with pandas
def a():
    #preprocess the file and save it
    #only if the file doesent already exists
    if not os.path.isfile(p_datafile):
        with open(p_datafile, 'w') as p_file:
            with open(datafile, 'r') as file:
                for line in file:
                    #add a 'tab' after each passwort frequentcy so it can be used as sep
                    processed_line = re.sub(r'^(\s+)([0-9]+)(\s)',r'\2\t', line)
                    p_file.write(processed_line)

    df = pd.read_csv(p_datafile, sep='\t', header=None,  names=['frequentcys', 'passwords'])
    return df
```

```
4750:340
844993:3 nan
4986233:1 num010975427
10807123:1 NAC1993lac
```

(b) *Describe the password set and passwords, e.g. size, distribution, password length, errors in the set. Each description should include at least the center (mean, median or mode), the dispersion (variance, range, percentile).*

-

```python
#find, write and correct errors and write some usefull data to file
def b(df):
     #find all NaN/ error rows
     nan_rows = df[df['passwords'].isna()]
     #save the index of error rows to list
     list_nan_index = nan_rows.index.tolist()
     #write error rows to a new file
     with open(error_file, 'w') as e_file:
          for index in list_nan_index:
               #find line in file
               line = linecache.getline(p_datafile, index+1)
               format_line = str(index) + ':' + line
               e_file.write(format_line)

     #correct errors in df (drop the rows)
     df = df.drop(df.index[list_nan_index])
     #reset index
     df.reset_index(inplace=True, drop=True)

     #start writing stats
     with open(stats_file, 'w') as file:
          file.write(df.describe().to_string() + '\n')
          file.write('median:\t' + str(df['frequentcys'].median()) + '\n')
          file.write('mode:\t' + str(df['frequentcys'].mode()) + '\n' )
          file.write('var:\t' + str(df.var()) + '\n' )
     return df
```

```
        frequentcys
count  1.434410e+07
mean   2.272893e+00
std    9.229819e+01
min    1.000000e+00
25%    1.000000e+00
50%    1.000000e+00
75%    1.000000e+00
max    2.907290e+05
median: 1.0
mode: 0    1
dtype: int64
var: frequentcys    8518.955389
dtype: float64
```

(c) *Take a look at the ten most frequent passwords. Give a password (creation) policy for them. Your policy can be informal or very formal.*

- The ten most frequent passwords are the following:

```
   frequentcys  passwords
0       290729     123456
1        79076      12345
2        76789  123456789
3        59462   password
4        49952   iloveyou
5        33291   princess
6        21725    1234567
7        20901    rockyou
8        20553   12345678
9        16648     abc123
```

- These passwords underline the following policy's:

1. 5-9 characters
2. characters can be from the alphabet [0-9] or [a,b-z]

(d) *There is no single password policy. Select all passwords which are 7 to 32 characters long, contain at least one digit and at least one upper case letter. We call this set P1.*

-
```python
#create passwordset P1
def d_e(df):
    #\A from beginning of the string till \z end: looks for a word with 7-32 chars
    a = '\A(?=\w{7,32}\Z)'
    # [^a-z]* not A-Z 0-many times and then one time A-Z: looks for at least one
    upper case char
    b = '(?=[^A-Z]*[A-Z])'
    # D* no digits for 0-many times and then one digit: looks for at least one
    digit
    c = '(?=\D*\d)'
    p1 = df.loc[df['passwords'].str.contains(a+b+c, regex=True)].copy()
    #reset index
    p1.reset_index(inplace=True, drop=True)
    return p1
```

(e) *P1 will no contain all passwords which match to our policy. How do we handle passwords which do not appear but match to our policy? You either extend P1 or not. Give the reasons for your decision.*

- We did not extend P1. The strength of a password policy is measured empirically with a large sample of passwords. We want to know how strong the password policy is in a real world scenario. In a real world scenario, the passwords are created by people. Therefore the provided password sample file is ideal, as it contains passwords of accounts from real people. This means that if we were to extend P1 with passwords generated by an algorithm, it would falsify our desired result, ”because passwords are not usually distributed uniformly”(quote from Lecture 3)

(f) *Calculate the probability of each password in P1.*

-
```python
#calculate the probability of each password in p1
def f(p1, sum_of_frequentcys):
    p1['probability'] = p1['frequentcys'] / sum_of_frequentcys
    return p1
```

```
        frequentcys                          passwords  probability
0               449                          Password1     0.000523
1               396                          PASSWORD1     0.000461
2               298                          PRINCESS1     0.000347
3               236                          BABYGIRL1     0.000275
4               225                          Princess1     0.000262
5               217                           JORDAN23     0.000253
6               207                           Passw0rd     0.000241
7               203                            LOVE123     0.000236
8               175                            A123456     0.000204
```

(g) *Calculate Shannon Entropy of P1.*

- We calculated the Shannon Entropy with the following formula from the lecture:
$H(X) = -\sum_{i=1}^{\mathbb{N}} Pr[X = x_i]log_2(Pr[X = x_i])$

```python
#calculate Shannon Entropy of p1
def g(p1, sum_of_frequentcys):
    shannon_entropy = (p1['probability'] * numpy.log2(p1['probability'])*-1).sum()
    return shannon_entropy
```

The result is 19.10637335027745

(h) *Assume an online adversary with unlimited guesses likes to break 200 accounts. Estimate the expected number of guesses.*

- For calculating the expected number of guesses we also used a formula from the lecture: $G(X) = \sum_{i=1}^{\mathbb{N}} i \cdot Pr[X = x_i]$

```python
#calculates expected num of guesses to crack k accounts
def h(p1, num_of_accounts_to_crack):
    #sort values by probability from high to low (i > i+1)
    p1.sort_values(by=['probability'], ascending = False, inplace = True)
    #reset index
    p1.reset_index(inplace=True, drop=True)
    #calc guesswork
    guesswork = (p1['probability']* (p1.index +1 )).sum()
    return num_of_accounts_to_crack*guesswork
```

The result is 57652515

(i) *Assume an online adversary likes to break at least 50 of 200 accounts. Estimate adversary's work.*

- We calculated the Shannon Entropy with the following formula from the lecture: $\sum_{i=1}^{\beta} Pr[X = x_i] \geq \frac{l}{k}$

```python
#calculates expected num of guesses to crack l accounts of k existing accounts
def i(p1):
    l_div_k = 50/200
    sum = 0
    beta = 0
    while sum < l_div_k:
        sum += p1['probability'][beta]
        beta += 1
    return beta
```

The result is 62693

(j) *Formalize your own password policy and repeat the above estimations.*

- Our passowrd policy's follows the following rules:
  1. 8-64 characters
  2. contains at least two digit
  3. contains at least two upper case letter
  4. contains at least one lower case letter
- Result for the above estimations:

  f) **Probability**

```
       frequentcys                                        passwords  probability
0               41                                        ABCabc123     0.000563
1               30                                        ABC123abc     0.000412
2               23                                        abc123ABC     0.000316
3               13                                        123qweASD     0.000179
4               12                                        abcABC123     0.000165
5               11                                        Sail2Boat3    0.000151
6               10                                      JamesBond007    0.000137
7               10                                        2Bornot2B     0.000137
8                9                                        ABCdef123     0.000124
```

  g) **Shanon Entropy** result is 16.036519282236434

  h) The adversary needs 6565494.616061121 guesses to **break 200 accounts**

  i) The adversary needs 14473 guesses to **break 50 of 200 accounts**

(k) *Compare the password policies and estimations.*

The main difference between p1 and our own password policy is that we require two capital letters and and two numbers. We thought that this would make the passwords created by users more secure. However, the Shannon entropy and the number of guesses for cracking accounts have decreased. Perhaps this is due to the lower password sample size, or it is because our password policy restricts the number of possible passwords and thus becomes weaker. However, we tend to assume the first.

Estimations	**P1**	**Own password policy**
Password sample size	858.415	72.772
Shanon Entropy	19	16
Guesses to break 200 accounts	57.652.515	6.565.494
Guesses to break 50 of 200 accounts	62.693	14.473

2 Project

(a) *Setup Docker on your machine (or on a virtual machine).*

(b) *Add a docker image for your project. You find an example in the resources folder.*

a) *Some use full docker commands:*

```
#lists all images
docker image ls
#lists all containers
docker ps -a

#stops all containers
docker stop $(docker ps -a -q)
#deletes all containers
docker rm $(docker ps -a -q)
#deletes all images
docker rmi --all --force

#builds images of docker_compose.yml
docker-compose build
#creates containers of docker_compose.yml in detached mode and overwrites old
    images
docker-compose up -d --force-recreate
#deletes all images and containers created by last docker-compose up call
docker-compose down
```

b) *Docker and docker-compose files:*

```
FROM python:3.8-alpine
ADD client.py /client/
WORKDIR /client/
```

```
FROM python:3.8-alpine
ADD client.py /client/
WORKDIR /client/
```

```yaml
# file from https://www.freecodecamp.org/news/a-beginners-guide-to-docker-how-to-create-a-client-server-side-with-docker-compose-12c8cf0ae0aa/
# A docker-compose must always start by the version tag.
# We use "3" because it's the last version at this time.
version: "3"

# You should know that docker-composes works with services.
# 1 service = 1 container.
# For example, a service maybe, a server, a client, a database...
# We use the keyword 'services' to start to create services.
services:
  # First service (container): the server.
  # Here you are free to choose the keyword.
  # It will allow you to define what the service corresponds to.
  # We use the keyword 'server' for the server.
  server:
    # The keyword "build" will allow you to define
    # the path to the Dockerfile to use to create the image
    # that will allow you to execute the service.
    # Here 'server/' corresponds to the path to the server folder
    # that contains the Dockerfile to use.
    build: server/

    # The command to execute once the image is created.
    # The following command will execute "python ./server.py".
    command: python ./server.py2

    # Remember that we defined in'server/server.py' 1234 as port.
    # If we want to access the server from our computer (outside the container),
    # we must share the content port with our computer's port.
    # To do this, the keyword 'ports' will help us.
    # Its syntax is as follows: [port we want on our machine]:[port we want to retrieve in the container]
    # In our case, we want to use port 1234 on our machine and
    # retrieve port 1234 from the container (because it is on this port that
    # we broadcast the server).
    ports:
      - 1234:1234

  # Second service (container): the client.
  # We use the keyword 'client' for the server.
  client:
    # docker run -i
    # activates std input
    stdin_open: true
    # docker run -t
    # activates std output
    tty: true

    # Here 'client/ corresponds to the path to the client folder
    # that contains the Dockerfile to use.
    build: client/

    # The command to execute once the image is created.
    # The following command will execute "python ./client.py".
    command: python ./client.py

    # The keyword 'network_mode' is used to define the network type.
    # Here we define that the container can access to the 'localhost' of the computer.
    network_mode: host

    # The keyword'depends_on' allows you to define whether the service
    # should wait until other services are ready before launching.
    # Here, we want the 'client' service to wait until the 'server' service is ready.
    depends_on:
      - server

    # starts this service as container 'scale' times
    scale: 3
```

(c) *Before using third party dependencies. We might should talk about them and potential threats. Read (or watch the video) the paper Small World with High Risks: A Study of Security Threats in the npm Ecosystem and list at least 3 risks using npm. TODO*

(d) *Is this relevant for Python and our project? Give a short statement. You might find some research about Python, too. TODO*